Remarks

Upon entry of the foregoing amendment, claims 1-3, 5-13, 22, 26-33, 35-37, and 45-56 are pending in the application, with claims 1, 22, 31, and 45 being the independent claims. Claims 1, 22, 31, 33, and 45 are sought to be amended. Support for the amendments of claims 1, 22, 31, and 45 can be found, for example, on page 30, lines 5-7, of the specification. Support for the amendment of claim 33 can be found, for example, on page 23, line 4, through page 24, line 20, of the specification. These changes are believed to introduce no new matter, and their entry is respectfully requested. Applicants believe that entry of these amendments after final are appropriate because they more clearly differentiate the claims from the cited references.

Based on the above amendment and the following remarks, Applicants respectfully request that the Examiner reconsider all outstanding objections and rejections and that they be withdrawn.

Rejections Under 35 U.S.C. § 103

The current Office Action states on page 4 that claims 1-3, 5-13, 22, 26-33, 35-37, and 45-56 are rejected under U.S.C. § 103 as allegedly being unpatentable over U.S. Pat. No. 5,862,066 to Rossin *et al.* (hereinafter, "Rossin") in view of Marc Olano, "A Programmable Pipeline for Graphics Hardware," PhD Dissertation, Department of Computer Science, University of North Carolina, Chapel Hill, April 1998 (hereinafter, "Olano"). Applicants respectfully traverse this rejection. Even if Rossin and Olano are combined for the sake of argument, such a combination still does not teach or suggest the claimed invention. At the very least, neither Rossin nor Olano, taken alone or in combination, teaches or suggests a floating point frame buffer as recited in the claimed invention.

Summary of the Claimed Invention

The present application discloses and claims a computer system and method for graphics processing that can operate in the floating point format throughout the various stages in the graphics processing pipeline. The floating point format can be used to perform geometric calculations and rasterization, as well as to store data in a frame buffer. The data can also be scanned out (or read) from the frame buffer in the floating point format for further graphics calculations or for display. As discussed in the Background section of the specification, although floating point graphics calculations were performed in previous systems (see page 3, lines 2-6, of the specification), floating point frame buffering and scan out was not done for various reasons. (See pages 4-8 of the specification.)

Summary of the Rossin Reference

Rossin describes a computer graphics system that includes a geometry accelerator, a rasterizer and a frame buffer. (See Rossin, col. 2, lines 13-15.) The Examiner concedes on page 5 of the Office Action that "Rossin fails to explicitly teach 'a floating point frame buffer.'"

Summary of the Olano Reference

The Olano dissertation describes an abstract pipeline on which to model interactive graphics machines for user-written procedures, such as procedural shading. The dissertation includes an introduction, a description of an abstract pipeline, and a description of the PixelFlow graphics hardware system. The document also describes surface shading and lighting stages on the PixelFlow system and describes an implementation of primitive and interpolation stages. The document also discusses user experience of those who have written shading procedures for use on the PixelFlow system and concludes by discussing areas of future research.

Olano's abstract pipeline is introduced in Chapter 2 and shown in Figure 2.1. According to Figure 2.1, the abstract pipeline includes the following stages: model, transform, primitive, interpolate, shade (and light), atmospheric, and image warp. According to Olano, "each stage can be implemented as a procedure." However, Olano also states that hardware systems may not be able to support every programmed stage. (See Olano, p. 16.) In Chapter 3, Olano maps the abstract pipeline to the PixelFlow graphics hardware system. The mapped PixelFlow system includes a host workstation, various rendering nodes (handling the modeling, transformation, primitive, and interpolation stages), various shading nodes (handling the shading, lighting, and atmospheric stages), and a frame buffer node (handling the image warping stage). This is shown in Figures 3.1, 3.2, and 3.3. There is no detailed discussion of the frame buffer node or usage of a frame buffer. There is also no detailed discussion involving the scanning or reading out data from the frame buffer for display. The remainder of the document appears to focus only on the shading/lighting and primitive/interpolation stages.

Chapter 4 of the Olano document discusses surface shading and the *pfman* shading language, which is said to be similar to the RenderMan shading language. Chapter 4 discusses data types used for shading, and specifically states that the floating point format is supported by the RenderMan, *pfman*, and Mendelbrot shading languages. However, Olano states that "[o]ur pixel processors do not support floating point in hardware, so every floating point operation is built from basic integer math operations" (see Olano, p. 69). Thus, Olano's shader's usage of the floating point format appears to be emulated in software. At the end of Chapter 4 (Olano, pp. 79-80), various rendering, shading, and frame buffering PixelFlow stages are shown in Figure 4.20. The frame buffer stage is described as handling "copying the incoming image pixels into the frame buffer, including any required warping." There is no discussion of data type in this description.

Chapter 5 appears to discuss the primitive and interpolation stages. In section 5.2.1 on page 86, two other graphics pipelines are introduced. Generally, they include the stages of transforming, clipping, rasterizing, interpolating, and shading/texturing (see Figure 5.2). Subsequent to shading/texturing is "display," which does not appear to be considered a programmable stage. There is no discussion of data type in this description.

In the conclusions and discussion of future research of Chapter 7, image warping is discussed in section 7.1.6 on page 118. That discussion states that "PixelFlow has a testbed interface for new procedural code to run on the frame buffer board. However, there is only a single procedural hook to handle both image warping and the operations to load pixels into the frame buffer itself. This latter code is non-trivial and make new versions of this stage difficult to write. For usable support of procedural image warping, this stage should be split into a procedural warping stage and a separate frame buffer stage, hidden from the user." This section does not discuss data types or detailed usage of the frame buffer.

Differences Between Claimed Invention and Teachings of Cited Reference(s)

The Examiner concedes on page 5 of the Office Action that "Rossin fails to explicitly teach 'a floating point frame buffer,'" but states that Olano teaches a floating point frame buffer. Applicants do not disagree that Olano's RenderMan shading language supports the floating point format generally. However, Applicants do not believe that Olano's emulated system teaches a true floating point frame buffer. For example, a true floating point frame buffer can read (or scan) floating point values out for display. An emulated floating point system, such as Olano's system, must convert values to fixed point format for display, as discussed in the paragraph starting on line 12 of page 4 of the present specification. Reading values out of the frame buffer for display is a hardware function. Since Olano states on page 69 that "[o]ur pixel processors do not support floating point in hardware...," Olano does not teach reading floating point values out for display and does not teach a true floating point frame buffer.

Discussion of Examiner Remarks

To support the assertion that Olano teaches a floating point frame buffer, the Examiner states on pages 2-3 and 5 of the Office Action that according to Olano (p. 59), "RenderMan has one representation for all numbers: floating point." The Examiner goes on to state that the RenderMan and Mandelbrot shading languages support the floating point format. Applicants do not disagree that the RenderMan and Mandelbrot (and *pfman*) shading languages support the floating point format generally. However, the passing of data to a frame buffer and on to display are not discussed in detail in the Olano dissertation, let alone the data types used in doing so. For example, the Examiner states on pages 3 and 5 of the

Office Action that the "...RenderMan hardware includes a frame buffer memory for rendering a frame of image during the rendering or shading in the image pipeline." Applicants respectfully disagree. The PixelFlow graphics hardware system (the RenderMan shading language is not hardware) includes a frame buffer node that handles the image warping stage. Image warping is not the equivalent of loading pixels into the frame buffer (see the Summary of the Olano document, above, with regard to Chapter 7). There is no discussion in the Olano document regarding the role of the shading languages (RenderMan, *pfman*, etc.) in loading the frame buffer or reading out of the frame buffer for display. Thus, it cannot be said that Olano teaches a floating point frame buffer.

In other examples on pages 3 and 5-6, the Examiner states "Olano further discloses [on page 27 that] the Mandelbrot shader (the prior art teaching) renders a frame of image having visible detail to the precision limits of the computations involved which is displayed in Fig. 4.7 wherein a frame of an image is rendered in floating point precision" and states "Olano discloses in page 58 of rendering a frame at 30 frames per second in a system with four shaders using the paging method in which four bytes ... of texture memory for every pixel in a 128 by 64 region ... takes 380 μs in which a frame of the image is rendered in the shader in the floating point format." As stated above, Applicants do not disagree that the RenderMan, Mandelbrot, and *pfman* shading languages support the floating point format generally. However, the details of the passing of data to a frame buffer and on to display are not suggested or discussed in the Olano document. Since Olano does not suggest or discuss the frame buffer in detail and states on page 69 that "[o]ur pixel processors do not support floating point in hardware...," it cannot be said that Olano teaches a floating point frame buffer.

The Examiner states on page 5 that "Olano teaches a floating point frame buffer and rasterization process, PixelFlow, which operates on the floating point format" and cites Figures 2.1, 3.1, 3.2, 3.4, and 5.2, and pages 68-79 and 102-104 for support. PixelFlow is a graphics hardware system which, alone, does not operate on the floating point format, according to Olano. Olano states on page 69 that "[o]ur pixel processors do not support floating point in hardware, so every floating point operation is built from basic integer math operations." Thus, the PixelFlow system, at most, emulates the floating point format in software. The Olano document discusses this in regard to shading calculations (mainly in Chapter 4). Since the Olano document does not discuss in detail the usage of the frame buffer, and since Olano states on page 69 that "[o]ur pixel processors do not support floating point in hardware...," it cannot be said that Olano teaches a floating point frame buffer.

Figures 2.1, 3.1, and 3.2 of Olano teach that there is a frame buffer node that handles the image warping stage. As stated above, image warping is not the equivalent of loading pixels into the frame buffer (see the Summary of the Olano Reference, above, with regard to Chapter 7). There is no detailed discussion in the Olano document regarding the loading or reading out from the frame buffer, or the data types used when doing so. Thus, Figures 2.1, 3.1, and 3.2 do not teach a floating point frame buffer. Figure 3.4 shows the components of a PixelFlow node, which includes a 'texture/frame buffer memory.' However, this memory is not discussed in detail. It is unclear whether this memory is considered the frame buffer, and usage of this memory in connection with the loading and reading out of the frame buffer is not discussed. Even if, for the sake of argument, such 'texture/frame buffer memory' is considered a frame buffer, no floating point frame buffer is taught or suggested. Figure 5.2 shows two other examples of graphics pipelines. One includes a shade stage followed by 'display,' and the other includes a texture stage followed by 'display.' Any stages directly related to the frame buffer appear to be missing from this figure and its applicable text. Notice also that 'display' in Figure 5.2 (as well as 'image pixels' in Figure 2.1, for example) is not in a box and assumed by the Applicants to not be a programmable stage.

Pages 68-79 of Olano discuss shading calculations in detail and include discussion of floating point versus fixed point formats. Frame buffering appears to be discussed briefly on pages 79-80. A frame buffer node is shown in Figure 4.20(a), and is discussed only by stating on page 80 that "the frame buffer stage handles copying the incoming image pixels into the frame buffer, including any required warping." The Olano document does not go into detail as to how this 'copying' is done or by using what data type. As discussed above, image warping is not the equivalent of loading pixels into the frame buffer (see the Summary of the Olano Reference, above, with regard to Chapter 7).

Pages 102-104 discuss buffer space with regard to the implementation of the primitive and interpolation stages. It is unclear how this buffer space relates to the frame buffer, if it relates at all.

The Office Action states on page 7 that "[Olano] also teaches handling a floating point or fixed point frame buffer which is a portion of the rasterization pipeline within the graphics rendering pipeline." Presumably in support of this assertion, the Examiner then states that "[t]he color values received by the pipeline are represented in a floating point format which includes a mantissa portion and an exponent portion" and cites page 100. Applicants respectfully disagree with these assertions. Since the Olano document does not discuss in detail the usage of the frame buffer, and since Olano states on page 69 that "[o]ur pixel processors do not support floating point in hardware...," it cannot be said that Olano teaches a floating point frame buffer. Pages 99-100 of Olano discuss interpolator functions. On page 99, it is stated that "...PixelFlow restricts varying shading parameters to only use fixed-point types...." Even if the floating point format was used for calculations made in the primitive/interpolation stage(s), this does not teach a floating point frame buffer.

On page 8, the Office Action states that "[r]e claims 5 and 48, Rossin and Olano disclos[sic] the floating point format is comprised of sixteen bits" citing "Rossin col. 1, lines 32-44 and Olano Fig. 2.1, 3.1, 3.2, 3.4, 5.2 and Page 68-79, 102-104." Applicants respectfully disagree. Rossin appears to teach a less than 16-bit floating point representation and a 32-bit floating point representation. Neither the cited figures nor pages 102-104 of Olano teach floating point representation. Page 69 of Olano shows 16 and 32-bit fixed point and 32-bit floating point, but does not show 16-bit floating point.

On page 9, the Office Action states "[r]e claim 31, Rossin and Olano disclose a computer system comprising a raster subsystem for performing a rasterization process, the rasterization process performed in a floating point format and a floating point frame buffer coupled to the raster subsystem for storing a plurality of floating point color values.... In other words, Rossin and Olano teach a typical computer graphics system include[sic] a geometry accelerator, a rasterizer and a frame buffer in a graphics pipeline." Applicants respectfully disagree. As conceded by the Examiner on page 5 of the Office Action, Rossin does not teach a floating point frame buffer. Furthermore, as discussed above, since the Olano document does not discuss in detail the usage of the frame buffer, and since Olano states on page 69 that "[o]ur pixel processors do not support floating point in hardware...," it cannot be said that Olano teaches a floating point frame buffer.

Also on page 9, the Office Action states "[r]e claims 32-33 and 35, Olano discloses the floating point color values are written to, read from (for display purposes), and stored in the frame buffer" Applicants respectfully disagree. As discussed above, since the Olano document does not discuss in detail the usage of the frame buffer, and since Olano states on page 69 that "[o]ur pixel processors do not support floating point in hardware...," it cannot be said that Olano teaches a floating point frame buffer that is written to, read from, and stores in the floating point format.

On page 10, the Office Action states that "[r]e claims 36-37, Olano discloses the floating point color values are comprised of 16 bits of data and the data are comprised of one sign bit, ten mantissa bits, and five exponent bits" citing "Olano Fig. 2.1, 3.1, 3.2, 3.4, 4.7, 5.2 and Page 59, 68-79, 102-104." Applicants respectfully disagree. Neither the cited figures nor pages 102-104 of Olano teach floating point representation. Page 69 of Olano shows 16 and 32-bit fixed point and 32-bit floating point, but does not show 16-bit floating point.

Based on the above analysis, independent claims 1, 22, 31, and 45 have been amended to more clearly differentiate the claims from the cited reference(s). Neither Rossin nor Olano, taken alone or in combination, teaches or suggests "a display screen coupled to the frame buffer for receiving the plurality of image values read out from the frame buffer in the floating point format..." as recited in amended claim 1, for example. Independent claims 22, 31, and 45, as amended, each include a similar recitation. Dependent claim 33 has been amended to differentiate between reading values out from the frame buffer for display purposes (as in independent claim 31) versus reading values out from the frame buffer for graphics manipulation processing. Thus, as amended, independent claims 1, 22, 31, and 45, and all the claims depending therefrom (claims 2-3, 5-13, 26-30, 32-33, 35-37, and 46-56), are allowable for at least the reasons discussed above. Therefore, reconsideration and withdrawal of the rejection of claims 1-3, 5-13, 22, 26-33, 35-37, and 45-56 are respectfully requested.

Conclusion

All of the stated grounds of objection and rejection have been properly traversed, accommodated, or rendered moot. Applicants therefore respectfully request that the Examiner reconsider all presently outstanding objections and rejections and that they be withdrawn. Applicants believe that a full and complete reply has been made to the outstanding Office Action and, as such, the present application is in condition for allowance. If the Examiner believes, for any reason, that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at the number provided.

Prompt and favorable consideration of this Amendment and Reply is respectfully requested.

Respectfully submitted,

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.

[illegible]

Michael V. Messinger
Attorney for Applicants
Registration No. 37,575

Date: August 3, 2005

1100 New York Avenue, N.W.
Washington, D.C. 20005-3934
(202) 371-2600

416234_1.DOC